Chapter 1

Parallel Sparse Solvers, Preconditioners, and Their Applications

1.1 Introduction

Systems of linear equations arise at the heart of many scientific and engineering applications. Their solutions often dominate the execution times. The coefficient matrices of these linear systems are often sparse; that is, most of the matrix entries are zeros. For realistic modeling and simulations, the linear systems are very large; some have millions of unknowns. As technology advances, there is a greater demand for extremely high-fidelity simulations. Consequently the systems are becoming even larger than before. To solve them, it is essential to exploit sparsity. Moreover, parallel computing is an essential tool to reduce the solution times and, in some cases, may offer the only possibility of solving the systems. There are two main classes of methods for solving sparse linear systems: direct methods and iterative methods. As we will see later, hybrid methods that combine direct and iterative schemes are gaining popularity.

This chapter will sample some of the most recent work on the parallel solution of large sparse linear systems. Section 1.2 deals with issues in parallel sparse direct methods and Section 1.3 discusses advances in iterative methods and preconditioning techniques. Hybrid methods that combine techniques in sparse direct methods and iterative schemes are considered in Section 1.4. Section 1.5 surveys recent activities on expert systems on sparse matrix computation. Finally, we conclude in Section 1.6 with some applications of sparse matrix computations.

1.2 Sparse Direct Methods

Direct methods, based on triangular factorizations, have provided the most robust and reliable means for solving sparse systems of linear equations $Ax = b$. The solutions are obtained after a finite number of operations. Fill is generally introduced into the factors during the factorization. That is, some entries that correspond to zeros in A become nonzero. Managing fill during factorization is probably as important as designing efficient algorithms for performing the factorization. The

solution process generally consists of three phases: analysis, factorization, and triangular solution. A good introduction to sparse matrix factorization can be found in [22, 30]. During the analysis phase, the matrix is analyzed to determine a (tentative) pivot sequence and perhaps set up the appropriate data structures required for the numerical factorization. This may involve permuting (or ordering) the rows and/or columns of the matrix, as well as constructing tools (such as elimination trees [49, 63] and assembly trees [25]) for efficient numerical factorization. Depending on the characteristics of the matrix, the analysis can be entirely symbolic (e.g., for symmetric positive definite matrices) or can rely on both symbolic and numeric information (e.g., for general unsymmetric matrices). During the numerical factorization, the information from the analysis is used in computing the factors L and U, where L is lower triangular and U is upper triangular. Modifications to the pivot sequence may be needed to maintain numerical stability. The final phase uses the triangular factors to compute the solution to the linear systems. Iterative refinement may be used to improve accuracy.

Permuting the rows and columns during the analysis phase is combinatorial in nature. It often involves graph algorithms, which is the subject of Chapter ??. The numerical factorization is often the most time consuming step. Consequently, it was the first task that was tackled when parallel computing became available. There is a lot of structure in the triangular factorization. In particular, it is often the case that some consecutive rows and columns of the triangular factors have similar sparsity structures[1] (this can happen because of fill and in finite-element applications, for example, where there is more than one variable at each node). Such a group of rows or columns is often referred to as a supernode [6, 47]. The use of supernodes allows dense matrix kernels to be exploited. State-of-the art sparse direct solvers use Level 3 BLAS [18] and LAPACK [4] routines as much as possible. Once a triangular factorization has been computed, the third phase is relatively straightforward. In a parallel setting, triangular solutions are communication bound; the amount of communication is high compared to the number of floating-point operations required, particularly when solving for a single (or small number of) right-hand sides. As the performance of parallel numerical factorization algorithms improves, parallel sparse triangular solutions can become the bottleneck, particularly when a sequence of right-hand sides has to be solved.

1.2.1 Parallel Sparse Matrix Factorizations

While there has been significant progress in the improvement of the performance of parallel sparse matrix factorizations over the years, sparse direct methods have been less popular for large-scale applications than iterative methods. One reason is the large storage requirement of the numerical factorization phase. Another is the amount of communication required by parallel factorizations. The communication overhead depends, to a large extent, on how the work is partitioned and assigned to the processors.

Li, Grigori, and Wang [35, 44] have recently carried out a careful study of

[1] Sparsity structure of a matrix refers to the locations of the nonzero elements in the matrix.

the numerical factorization phase of SuperLU_DIST [43], a state-of-the-art direct solver for sparse unsymmetric systems. The L and U factors are represented in SuperLU_DIST as 2D block matrices, and are distributed in a 2D block-cyclic fashion on a 2D process grid. By examining the sparsity structure of L and U, and the communication pattern of the algorithm, Li *et al.* have derived a theoretical bound on the parallel efficiency of the algorithm. They have demonstrated that this bound can be used to predict the performance for any sparse matrix, when the sparseness is measured with respect to the underlying machine parameters, such as floating-point speed, network latency, and bandwidth. The theoretical efficiency bound may be used to determine the most critical hardware parameters that need to be improved to enhance performance for the class of problems under consideration.

Li *et al.* have analyzed the efficiency of SuperLU_DIST using increasing numbers of processors with increasing problem sizes. They conclude that for matrices satisfying a certain relation between their size and their memory requirements (namely workload F proportional to $nnz(L+U)^{3/2}$),[2] the factorization algorithm is scalable with respect to memory use. This relation is satisfied by matrices arising from 3D model problems. For these problems, the efficiency is essentially constant when the number of processors increases while the memory requirement per processor is constant. However, for 2D model problems, the algorithm does not scale with respect to memory use. Li *et al.* have validated their results on an IBM Power3 parallel machine at the National Energy Research Scientific Computing (NERSC) Center. It appears that load imbalance and an insufficient work relative to the communication overhead are the main sources of inefficiency on a large number of processors.

In a separate study, Guermouche and L'Excellent [36] have considered memory requirements in a parallel implementation of a multifrontal algorithm [25]. They distinguish between storage for the factors (referred to as static) and that for the frontal matrices and the contribution blocks (referred to as dynamic). The static storage grows as the factorization proceeds, while the size of the dynamic storage varies depending on the structure of the assembly tree. In the factorization of a frontal matrix, processors are selected based on their current memory usage. In addition, Guermouche and L'Excellent have taken memory usage into consideration when parallel tasks are scheduled. They have demonstrated that overall memory requirements can be improved using a combination of task distribution and management strategies, without greatly degrading the factorization times.

Another factor that affects the performance of sparse matrix factorization is data mapping. It is well known that parallelism can be identified using a tree structure (e.g., an elimination tree for Cholesky factorization [49, 63] and an assembly tree for multifrontal methods [25]). The *proportional heuristic* [55, 56], which is a generalization of the subtree-to-subcube mapping [32], has typically been used to map the data and computation to processors. However, for sparse systems from finite-element methods on complex domains, the resulting assignments can exhibit significant load imbalances. Malkowski and Raghavan [50] have developed a multi-pass mapping scheme to reduce such load imbalances. The multi-pass mapping

[2] $nnz(M)$ denotes the number of nonzeros in the matrix M.

scheme combines the proportional heuristic with refinement steps to adjust the loads on the processors. For a test suite of large sparse matrices, they have demonstrated that the new mapping scheme is indeed effective in producing assignments that substantially improve the load balance among processors.

1.2.2 Ordering to Increase Parallelism

It is well known that for sparse direct methods, the sparsity structure of the triangular factors depends drastically on the ordering of rows and columns [22, 30]. A lot of work has been done in developing fill-reducing orderings. Examples of fill-reducing ordering algorithms include Markowitz scheme [51], the minimum degree algorithm [31, 58], and the nested dissection algorithm [29, 46].

It is generally believed that the sparser the triangular factors, the more independent the columns/rows, and hence the higher the degree of parallelism that is available. However, one can look at the ordering problem differently.

Duff and Scott [26] have considered using graph partitioning techniques to permute a unsymmetric matrix A into a *singly bordered block diagonal* form:

$$A \rightarrow \begin{bmatrix} A_1 & & & B_1 \\ & A_2 & & B_2 \\ & & \ddots & \vdots \\ & & A_q & B_q \end{bmatrix},$$

where the diagonal blocks A_i, $1 \leq i \leq q$, need not be square. A sparse direct solver can be applied to the diagonal blocks A_i independently and in parallel. Since A_i is in general rectangular, its factorization leads to a lower trapezoidal factor and an upper triangular factor:

$$A_i = \begin{bmatrix} L_i \\ \tilde{L}_i \end{bmatrix} U_i,$$

where L_i is square and lower triangular. By putting $\tilde{L}_i U_i$, $1 \leq i \leq q$, together, one obtains the following partitioning of A:

$$A \rightarrow \begin{bmatrix} L_1U_1 & & & & C_1 \\ & L_2U_2 & & & C_2 \\ & & \ddots & & \vdots \\ & & & L_qU_q & C_q \\ \tilde{L}_1U_1 & & & & D_1 \\ & \tilde{L}_2U_2 & & & D_2 \\ & & \ddots & & \vdots \\ & & & \tilde{L}_qU_q & D_q \end{bmatrix}.$$

Here, each B_i, $1 \leq i \leq q$, has been partitioned according to the row dimensions of L_i and $\tilde{L}_i$. That is,

$$B_i = \begin{bmatrix} C_i \\ D_i \end{bmatrix}.$$

This is termed a *stabilized doubly bordered block diagonal* form. The border is larger than that for a doubly bordered form obtained directly using a graph partitioning algorithm but the key advantage is that pivots are chosen stably from the diagonal blocks. For efficiency, it is desirable for B_i to have as few columns as possible and to be of a similar size. This is a combinatorial problem. Duff and Scott [26] have developed a number of coarse-grained parallel direct solvers based on preordering A to bordered form. These are available in the HSL mathematical software library [24].

1.2.3 Out-of-core Methods

One of the primary reasons why sparse direct methods may not be popular among extremely large-scale applications is the amount of storage required for the fill in the triangular factors. A number of codes are available that aim to alleviate the storage bottleneck by incorporating out-of-core techniques [21, 48]. These involve writing part of the matrix factors that have been computed but no longer needed during the factorization to secondary storage (e.g., disks). Only a small portion of the matrix factors is kept in the main memory. As a result, the main memory requirement can be much reduced. Almost all out-of-core implementations have been based on frontal and multifrontal methods [25, 40]. Oblio, an object-oriented library for solving sparse linear systems of equations by direct methods designed and developed by Dobrian and Pothen [17], is designed to handle memory accesses at two levels: in-core and out-of-core. At the in-core level the supernodal nature of the computation provides support for Level 3 dense linear algebra kernels. Most of the time these kernels translate directly into BLAS/LAPACK [4, 18] calls but there are few cases that need to be customized using recursion or blocking. At the out-of-core level, Oblio can store the matrix factors on disk.

1.3 Iterative Methods and Preconditioning Techniques

An iterative method for solving a linear system is based on the generation of a sequence of approximations to the solution. Whether the sequence will converge to the solution depends a great deal on the matrix, the starting guess, and the choice of iterative method. There are many iterative schemes available. Conjugate gradient (CG), minimal residual (MINRES), generalized minimal residual (GMRES), biconjugate gradient (BCG), conjugate gradient squared (CGS), biconjugate gradient stabilized (BiCGSTAB), quasi-minimal residual (QMR), and Chebyshev iteration are some of the most popular schemes [59, 67].

Iterative methods are very popular in large-scale applications because their storage requirements are not as severe as those in sparse direct methods. In most cases, just the original matrix and a few vectors need be stored; there is no fill to worry about. Moreover, the kernels are mostly inner products and sparse matrix-vector multiplications, which are easier to implement and parallelize than sparse direct methods. However, there are few operations to perform so that the efficiency

of parallel iterative methods tends to be poor.

The spectral properties of the iteration matrix play a very important role in the rate of convergence. One can change the spectral properties by *preconditioning* the linear system. More specifically, instead of solving $Ax = b$, one can solve

$$PAQQ^{-1}x = Pb,$$

where P and Q are referred to as the left and right preconditioners, respectively. Constructing good preconditioners has been the subject of much research during the last decade. There are many possibilities. Well-known ones include Jacobi, SSOR, incomplete factorization, approximate inverses, and multilevel preconditioning [59, 67]. The incorporation of the preconditioning step complicates the implementation somewhat, but can potentially significantly improve the convergence rate and may be the only way that convergence can be achieved.

1.3.1 Improving the Performance of Iterative Solvers

Matrix-vector multiplication is an important part of many iterative solvers. Its implementation can drastically affect the performance of an entire iterative solver. For sparse problems, this kernel has very little data reuse and a high ratio of memory operations to floating-point operations. Recently, Kaushik and Gropp [41] have analyzed the performance based on memory bandwidth, instruction issue rate, and the fraction of floating-point workload. Performance bounds based on these parameters provide a good estimate of achievable performance. To get better performance, they recommend multiplying the sparse matrix by more than one vector whenever possible. They compare the actual performance of this kernel with the derived performance bounds on scalar processors and a vector processor (a Cray X1). They observe that the performance is memory bandwidth limited on most scalar and vector processors. Even though memory bandwidth is huge on vector processors (as compared to most scalar machines), sparse matrix-vector multiplication is still memory bandwidth limited: they report that only 23% of the machine peak is possible under ideal situations for the Cray X1's. Similar work can be found in [39, 66, 68].

The work of Baker, Dennis, and Jessup [7] supports the finding of Kaushik and Gropp. Their work attempts to optimize the performance of iterative methods, and is based on a variant of the GMRES algorithm, in which the standard Krylov subspace is augmented with approximations to the errors from previous restart cycles. They have focused on a block implementation of this variant of GMRES that uses a "multivector" data structure. The use of this data structure allows groups of vectors to be interlaced together in memory. They have examined the memory characteristics of the block variant of GMRES, and have found that the use of a multivector data structure reduces data movement versus a non-interlaced or conventional approach. The advantage of the multivector data structure is not limited to the matrix-vector multiplication, but reduces data movement for all components of the iterative solver. They have demonstrated using numerical experiments that reductions in data movement do lead to reductions in execution time

1.3.2 Preconditioning Techniques

As already noted, preconditioning is crucial for the convergence of iterative methods. Much of the research into iterative methods in recent years has focused on the development of efficient and robust preconditioning techniques. Hénon and Saad [60] have proposed computing parallel incomplete LU factorizations in a hierarchical way. An incomplete factorization is basically an approximate factorization of the matrix, in which some of the fill entries in the exact factorization are discarded according to some prescribed criteria [52]. The Parallel Hierarchical Interface Decomposition ALgorithm (PHIDAL), proposed by Hénon and Saad, exploits Schur complements based on independent sets of "interfaces". The idea is reminiscent of the so-called "wirebasket" techniques of domain decomposition [64]. It can also be considered as a variation and an extension of the parallel Algebraic Recursive Multilevel Solver (pARMS) of [45]. As the name implies, interfaces (or separators) are constructed in a hierarchical manner. Once the hierarchical interface decomposition is defined, the Gaussian elimination process proceeds by levels: nodes in the first level are eliminated first, followed by those in the second, and so on. Drop tolerance strategies are defined so as to limit fill. More specifically, any nonzero entries that are smaller than a prescribed drop tolerance will be discarded. Hénon and Saad have reported that for system arising from the solution of Poisson equations the iteration and factorization times scale well using up to 256 processors, and are excellent for a more general unstructured problem.

1.3.3 Ordering for Incomplete Factorization

One of the most popular approaches to obtain a preconditioner is to use an incomplete factorization of the original coefficient matrix. As in direct methods, the ordering of the rows and columns can affect the number of fill entries that come up in an incomplete factorization. Since some of these fill entries are discarded, different row/column orderings will generally result in incomplete factors that can behave quite differently as preconditioners.

The most common way to order the rows and columns in an incomplete factorization is to preorder the coefficient matrix using ordering algorithms that have been designed to reduce fill in sparse direct methods (such as the Cuthill-McKee algorithm [13], the minimum degree algorithm [31, 58], and the nested dissection algorithm [29, 46]). Then incomplete factorization is applied to the preordered matrix.

In [53], Munksgaard ordered the rows and columns of a symmetric matrix while he was computing the incomplete factorization. The ordering was based on the minimum degree heuristic. In [15, 16], D'Azevedo, Forsyth, and Tang investigated the use of greedy heuristics (such as minimum degree and minimum deficiency [65]) to order the rows and columns while computing the incomplete factorization. However, numerical information (such as the norm of the discarded fill) was incorporated into the metrics used to select the pivot row and pivot column at each step. While the quality of the resulting incomplete factors (as preconditioners) was quite good, the orderings were quite expensive to compute.

Recently, Lee, Raghavan, and Ng [42] have considered an approach similar to Munksgaard's for computing incomplete Cholesky factorization. That is, the minimum degree algorithm and the numerical computation interleave throughout the factorization process. The minimum degree metric is applied to the sparsity pattern of the matrix resulted from incomplete Cholesky factorization. Lee *et al.* refer to this as the *interleaved minimum degree* strategy. However, more sophisticated techniques are used to modify the diagonal when the matrix loses positive definiteness. Preliminary results have showed that, as preconditioners the conjugate gradient iterations, the incomplete factors produced by the new schemes often exhibit improved convergent properties.

1.4 Hybrid Direct/Iterative Techniques

A relatively new but promising area of research in the solution of sparse linear systems is to combine techniques developed for sparse direct methods and iterative methods. Most of the investigations so far have been focused on the exploitation of techniques developed for sparse direct methods in computing incomplete factorizations. In particular, Ng and Raghavan [54, 57] have considered the use of dense matrix kernels in incomplete Cholesky factorization. More specifically, additional constraints are imposed so that dense submatrices are created in the incomplete Cholesky factors. Also, as was mentioned in Section 1.3.3, ordering techniques for sparse direct methods are often used in the context of computing incomplete factorization, which is then employed as preconditioners.

1.4.1 Applying Direct and Iterative Strategies to Partitioned Matrices

The singly bordered block diagonal form introduced in Section 1.2.2 provides a different approach to the problem. Duff *et al.* [23] have proposed combining direct and iterative methods to solve these large sparse unsymmetric equations. As in Section 1.2.2, a direct method is used to partially factorize local rectangular systems. Since the Schur complement matrix is generally quite dense, it can be advantageous to apply an iterative method to solve the subsystem instead of using direct methods.

Another possibility currently being examined is extracting a block diagonal matrix from the bordered form

$$\begin{bmatrix} L_1U_1 & & & & & \\ & L_2U_2 & & & & \\ & & \ddots & & & \\ & & & L_qU_q & & \\ & & & & D_1 & \\ & & & & D_2 & \\ & & & & \vdots & \\ & & & & D_q & \end{bmatrix},$$

and using it as a preconditioner for an iterative scheme on the whole linear system. The advantage is that the matrix $D^T = \begin{bmatrix} D_1^T & D_2^T & \dots & D_q^T \end{bmatrix}$ should be sparser than the Schur complement matrix. However, D can be singular, so some modifications may be needed in order to obtain a stable factorization of D.

1.4.2 Mixing Direct and Iterative Methods

Non-overlapping domain decomposition is a classical approach to solving large-scale PDE problems on parallel distributed computers. This technique often leads to a reduced linear system defined on the interface between the subdomains. Giraud, Mulligan, and Rioual [34] have recently considered iterative as well as direct solution techniques that exploit features of the MUMPS package [3]. Their block preconditioner consists of assembling the local Schur complement matrices associated with each subdomain and can be viewed as an additive Schwarz preconditioner. Efficient implementation relies on the capability of MUMPS to compute for each subdomain the local Schur complement matrix as well as the factorization of the local Dirichlet problem. The local Schur complement matrices are factorized by a final parallel application of MUMPS. This step uses the ability of MUMPS to process matrices given in a distributed format. Giraud *et al.* also consider the solution of the original linear system defined on the complete domain (i.e. not only its restriction to the interface between the subdomains), again using a parallel application of MUMPS and its ability to handle distributed input matrices.

These approaches have been integrated into a mixed finite-element code for the simulation of semiconductor devices in 2D where the nonlinear equations are solved using a Newton-Raphson scheme. Experiments have been performed on up to 32 processors for the simulation of a mosfet device. Results show that the iterative approach outperforms a classical direct substructuring technique [33]. Preliminary results for 3D heterogeneous diffusion problems with 9×10^6 degrees of freedom also indicate good scalability of the iterative solver on up to 350 processors.

Another approach that is becoming popular is to add the capability of computing an incomplete factorization using a sparse direct solver. The incomplete factorization can be used as a preconditioner for solving a sparse linear system using an iterative method. The resulting package can be considered a hybrid approach for handling very large problems. An example can be found in PaStiX, a library for solving sparse linear systems on clusters of SMP nodes using supernodal algorithms [37, 38].

1.4.3 Row-projection Methods

The Block Cimmino method [5, 12, 27], which is a block row-projection method for the solution of large sparse general systems of equations, is another hybrid method. The rows of the system matrix A are first partitioned into blocks. The block Cimmino method projects the current iterate simultaneously onto the manifolds corresponding to the blocks of rows and then takes a convex combination of the resulting vectors.

Drummond [19, 20] has investigated the choice of row partitionings. In par-

ticular, he has considered two different preprocessing strategies that are applied to the matrix AA^T. The goal is to find permutations that transform AA^T into a matrix with a block tridiagonal structure. Doing this provides a natural partitioning of the linear system for row projection methods because these methods use the normal equations to compute their projections. Therefore, the resulting natural block partitioning should improve the rate of convergence of block row projection methods and as block Cimmino and block Kaczmarz. The first preprocessing strategy produces a two-block partition, which naturally induces parallelism and improves the convergence of the block Cimmino method. The second strategy works on matrices with narrower bands, resulting in more blocks that can be processed in parallel. While the latter strategy exploits parallel computing better and could deliver two-block partitions, it is also more dependent on the parameters used to narrow the band of the matrix, the sizes of the resulting blocks, and the interconnection mechanisms between the processing elements.

1.5 Expert Approaches to Solving Sparse Linear Systems

From the above discussion, it is clear that many algorithms are currently available for solving sparse systems of linear equations. They come in different flavors: direct methods, iterative methods, and hybrids of the two. These methods represent different trade-offs with respect to metrics such as robustness, scalability, and execution time. Their performance can vary dramatically depending on the characteristics of a given linear system. Attempts are now being made to develop expert approaches to ease the selection of methods and/or to improve the robustness of the solution process.

One such example is the Grid-TLSE project [2, 11]. The goal of the project is to provide one-stop shopping for users of sparse direct methods. To accomplish this goal, the project will maintain a collection of sparse direct solvers and test matrices, as well as a database of performance statistics. A user will be able to interrogate the database for information concerning the sparse direct solvers or matrices that are available in the collection. The user will also be able to perform comparative analysis of selected solvers using user-supplied matrices or specific matrices from the matrix collection available in the GRID-TLSE project.

Another example is the work of Bhowmick *et al.* [10]. The purpose of this project is to improve the performance of large-scale simulations. The authors have developed *adaptive* linear solvers, which use heuristics to select a solution method to cope with the changing characteristics of linear systems generated at different stages of an application. They have also developed *composite* linear solvers, which apply a sequence of methods to the same linear system to improve reliability. Preliminary experiments have demonstrated that both approaches can significantly decrease execution times in some nonlinear PDE-based simulations from computational fluid dynamics.

1.6 Applications

As we have indicated in Section 1.1, there is an abundance of applications that rely on sparse linear solvers. In many cases, the matrix can become very large because of the need to perform high-fidelity modeling and simulations. In this section, we consider two applications: circuit simulations and structural dynamics.

Circuit simulations give rise to very large sparse linear systems, which can be highly ill-conditioned and therefore a challenge to solve. Recently, both direct and iterative methods of solution have been considered. Davis and Stanley [14] have considered unsymmetrically permuting the matrices to upper triangular form, then applying a symmetric minimum degree algorithm is to the diagonal blocks, which are then factorized using a sparse *LU* factorization with numerical pivoting. Sosonkina and Saad have employed pARMS [45, 61], a suite of parallel iterative accelerators and multilevel preconditioners for the solution of general sparse linear systems, to solve circuit simulation problems. Partitioning techniques and local ordering strategies play an important role in the solution process. These techniques also are relevant in the work of Basermann *et al.* [8]. Incomplete *LU* factorizations [8] and approximate inverses [62] have also been used as preconditioners for iterative methods of solution.

Salinas [9], a massively parallel structural dynamics code developed at Sandia National Laboratories, is a big user of parallel sparse linear solvers. During the early development of Salinas, FETI-DP (Dual-Primal Finite Element Tearing and Interconnecting) [28] was the primary linear solver used. Within each domain, sparse direct solvers are employed. Iterative linear solvers are used to handle the interface variables. Recently, Salinas developers have implemented an abstract interface to linear solvers, which has opened the door to two additional iterative linear solver packages: Prometheus (an algebraic multigrid package) [1], and CLASP (a multigrid solver developed by Clark Dohrmann at Sandia National Laboratories).

Acknowledgments

The author appreciated the input provided by Leroy Anthony Drummond, Luc Giraud, Laura Grigori, Dinesh Kaushik, Xiaoye Li, and Padma Raghavan. Without the input, it would be simply impossible to put the overview together. The author also would like to express his special thanks to Iain Duff and Jennifer Scott, who read an early draft and provided much feedback to improve the presentation.

This work was supported by the Director, Office of Science, U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Bibliography

[1] M. F. Adams, *Multigrid Equation Solvers for Large Scale Nonlinear Finite Element Simulations*, ph.d. dissertation, University of California, Berkeley, 1998.

[2] P. Amestoy, I. Duff, L. Giraud, J.-Y. L'Excellent, and C. Puglisi, *Grid-tlse: A web site for experimenting with sparse direct solvers on a computational grid*, in Eleventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, CA, 2004.

[3] P. Amestoy, I. Duff, J. Koster, and J.-Y. L'Excellent, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 15–41.

[4] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide, Third Edition*, SIAM, Phildelphia, PA, 1999.

[5] M. Arioli, I. Duff, J. Noailles, and D. Ruiz, *Block cimmino and block ssor algorithms for solving linear systems in a parallel environment*, Tech. Rep. TR/89/11, CERFACS, Toulouse, France, 1989.

[6] C. Ashcraft, R. Grimes, J. Lewis, B. Peyton, and H. Simon, *Progress in sparse matrix methods for large linear systems on vector supercomputers*, Internat. J. Supercomp. Appl, 1 (1987), pp. 10–30.

[7] A. Baker, J. Dennis, and E. Jessup, *An efficient block variant of GMRES*. Submitted, 2003.

[8] A. Basermann, I. Jaekel, M. Nordhausen, and K. Hachiya, *Parallel iterative solvers for sparse linear systems in circuit simulation*, Future Generation Comp. Syst., 21 (2005), pp. 1275–1284.

[9] M. Bhardwaj, K. Pierson, G. Reese, T. Walsh, D. Day, K. Alvin, J. Peery, C. Farhat, and M. Lesoinne, *Salinas: A scalable software for high-performance structural and solid mechanics simulations*, in SC2002, 2002.

[10] S. Bhowmick, P. Raghavan, and K. Teranishi, *A combinatorial scheme for developing efficient composite solvers*, in Lecture Notes in Computer Science, P. Sloot, C. Tan, J. Dongarra, and A. Hoekstra, eds., vol. 2330 of Computational Science-ICCS 2002, Springer Verlag, 2002.

[11] E. Caron, F. Desprez, M. Dayd; A. Hurault, and M. Pantel, *On deploying scientific software within the grid-tlse project*, Lectures in Computing Letters, 1 (2005).

[12] G. Cimmino, *Calcolo approssimato per le soluzioni dei sistemi di equaziono lineari*, in Ricerca Sci. II, vol. 9, I, 1938, pp. 326–333.

[13] E. Cuthill, *Several strategies for reducing bandwidth of matrices*, in Sparse Matrices and their Applications, D. J. Rose and R. A. Willoughby, eds., New York, 1972, Plenum Press.

[14] T. Davis and K. Stanley, *Sparse lu factorization of circuit simulation matrices*, in Eleventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, CA, 2004.

[15] E. D'Azevedo, P. Forsyth, and W. Tang, *Ordering methods for preconditioned conjugate gradients methods applied to unstructured grid problems*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 944–961.

[16] ——, *Towards a cost effective high order ilu preconditioner*, BIT, 32 (1992).

[17] F. Dobrian and A. Pothen, *OBLIO: Design and performance*, in State of the Art in Scientific Computing, Lecture Notes in Computer Science, 3732, J. Dongarra, K. Madsen, and J. Wasniewski, eds., Springer Verlag, 2005, pp. 758–767.

[18] J. J. Dongarra, J. D. Croz, S. Hammarling, and I. Duff, *A set of level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16 (1990), pp. 1–17.

[19] L. Drummond, *Block iterative methods for the solution of large sparse linear systems in heterogenous distributed computing environments*, PhD thesis, CERFACS, 1995.

[20] L. Drummond, I. Duff, and D. Ruiz, *Partitioning strategies for the block cimmino algorithm*, in Eleventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, CA, 2004.

[21] I. Duff, *Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core*, SIAM J. Sci. Stat. Comput., 5 (1984), pp. 270–280.

[22] I. Duff, A. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, England, 1987.

[23] I. Duff, G. Golub, J. Scott, and F. Kwok, *Combining direct and iterative methods to solve partitioned linear systems*, in Eleventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, CA, 2004.

[24] I. Duff, R. Grimes, and J. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Softw., 15 (1989), pp. 1–14.

[25] I. Duff and J. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Softw., 9 (1983), pp. 302–325.

[26] I. S. Duff and J. A. Scott, *Stabilized bordered block diagonal forms for parallel sparse solvers*, Parallel Computing, 31 (2005), pp. 275–289.

[27] T. Elfving, *Block-iterative methods for consistent and inconsistent linear equations*, Numer. Math., 35 (1980), pp. 1–12.

[28] C. Farhat, M. Lesoinne, P. LeTallec, K. Pierson, and D. Rixen, *Feti-dp: a dual-primal unified feti method - part i: A faster alternative to the two-level feti method*, Intern. J. Numer. Methods in Engineering, 50 (2001), pp. 1523–1544.

[29] J. George, *Nested dissection of a regular finite element mesh*, SIAM J. Num. Anal., 10 (1973), pp. 345–363.

[30] J. George and J. W.-H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.

[31] ——, *On the evolution of the minimum degree algorithm.* (To appear in SIAM Review), 1989.

[32] J. George, J. W.-H. Liu, and E. G.-Y. Ng, *Communication results for parallel sparse Cholesky factorization on a hypercube*, Parallel Computing, 10 (1989), pp. 287–298.

[33] L. Giraud, A. Marrocco, and J. Rioual, *Iterative versus direct parallel substructuring methods in semiconductor device modelling*, Num. Linear Algebra and Appl, 12 (2005), pp. 33–53.

[34] L. Giraud, S. Mulligan, and J. Rioual, *Algebraic preconditioners for the solution of schur complement systems*, in Eleventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, CA, 2004.

[35] L. Grigori and X. S. Li, *Performance analysis of parallel right-looking sparse lu factorization on two dimensional grids of processors*, in PARA'04 Workshop on State-of-the-art in Scientific Computing, June 20-23, 2004, Copenhagen, Denmark, 2004.

[36] A. Guermouche and J.-Y. L'Excellent, *Constructing memory-minimizing schedules for multifrontal methods*, ACM Transactions on Mathematical Software, (2005). to appear.

[37] P. Hénon, F. Pellegrini, P. Ramet, J. Roman, and Y. Saad, *High performance complete and incomplete factorizations for very large sparse systems by using Scotch and PaStiX softwares*, in Eleventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, CA, 2004.

[38] P. Hénon, P. Ramet, and J. Roman, *PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems*, Parallel Computing, 28 (2002), pp. 301–321.

[39] E.-J. Im and K. Yelick, *Optimizing sparse matrix computations for register reuse in SPARSITY*, Lecture Notes in Computer Science, 2073 (2001), pp. 127–136.

[40] B. Irons, *A frontal solution program for finite element analysis*, Int. J. Num. Meth. Engng., 2 (1970), pp. 5–32.

[41] D. Kaushik and W. Gropp, *Optimizing sparse matrix-vector operations on scalar and vector processors*, in Eleventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, CA, San Francisco, CA, 2004.

[42] I. Lee, P. Raghavan, and E. G. Ng, *Effective preconditioning through ordering interleaved with incomplete factorization*, SIAM J. Matrix Anal. Appl., (2006).

[43] X. S. Li and J. W. Demmel, *SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems*, ACM TOMS, 29 (2003), pp. 110–140.

[44] X. S. Li and Y. Wang, *Performance evaluation and enhancement of SuperLU_DIST 2.0*, Tech. Rep. LBNL-53624, Lawrence Berkeley National Laboratory, 2003.

[45] Z. Li, Y. Saad, and M. Sosonkina, *parms: A parallel version of the algebraic recursive multilevel solver*, Numerical Linear Algebra with Applications, 10 (2003), pp. 485–509.

[46] R. Lipton, D. Rose, and R. Tarjan, *Generalized nested dissection*, SIAM J. Num. Anal., 16 (1979), pp. 346–358.

[47] J. Liu, E. Ng, and B. Peyton, *On finding supernodes for sparse matrix computations*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 242–252.

[48] J. W.-H. Liu, *An adaptive general sparse out-of-core Cholesky factorization scheme*, SIAM J. Sci. Stat. Comput., 7 (1987), pp. 585–599.

[49] ——, *The role of elimination trees in sparse factorization*, Tech. Rep. CS-87-12, Dept. of Computer Science, York University, 1987. (to appear in SIAM J. Matrix Anal.).

[50] K. Malkowski and P. Raghavan, *Multi-pass mapping schemes for parallel sparse matrix computations*, in Proceedings of ICCS 2005: 5th International Conference on Computational Science, Lecture Notes in Computer Science, Number 3514, V. Sunderam, G. van Albada, P. Sloot, and J. Dongarra, eds., Springer Verlag, 2005, pp. 245–255.

[51] H. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Sci., 3 (1957), pp. 255–269.

[52] J. Meijerink and H. vander Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comput., 31 (1977), pp. 148–162.

[53] N. Munksgaard, *Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients*, ACM Trans. Math. Softw., 6 (1980), pp. 206–219.

[54] E. G. Ng and P. Raghavan, *Towards a scalable hybrid sparse solver*, Concurrency: Practice and Experience, 12 (2000), pp. 53–68.

[55] A. Pothen and C. Sun, *A mapping algorithm for parallel sparse cholesky factorization*, SIAM J. Sci. Comput., 14 (1993), pp. 1253–1257.

[56] P. Raghavan, *Parallel Sparse Matrix Factorization: QR and Cholesky Decompositions*, PhD thesis, The Pennsylvania State University, 1991.

[57] P. Raghavan, K. Teranishi, and E. G. Ng, *A latency tolerant hybrid sparse solver using incomplete cholesky factorization*, Numerical Linear Algebra and Applications, 10 (2003), pp. 541–560.

[58] D. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. C. Read, ed., Academic Press, 1972, pp. 183–217.

[59] Y. Saad, *Iterative Methods for Sparse Linear Systems, Second Edition*, SIAM, Philadephia, PA, 2003.

[60] Y. Saad and P. Henon, *PHIDAL: A parallel multilevel linear solver based on a hierarchical interface decomposition*, in Eleventh SIAM Conference on parallel Processing for Scientific Computing, San Francisco, CA, 2004.

[61] Y. Saad and M. Sosonkina, *parms: a package for solving general sparse linear systems of equations*, in Parallel Processing and Applied Mathematics, R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Wasniewski, eds., vol. 2328 of Lecture Notes in Computer Science, Berlin, 2002, Springer-Verlag, pp. 446–457.

[62] O. Schenk, *Recent advances in sparse linear solver technology for semiconductor device simulation matrices*, in Eleventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, CA, 2004.

[63] R. Schreiber, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math. Softw., 8 (1982), pp. 256–276.

[64] B. Smith, P. Bjorstad, and W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, 1996.

[65] W. Tinney and J. Walker, *Direct solution of sparse network equations by optimally ordered triangular factorization*, Proc. IEEE, 55 (1967), pp. 1801–1809.

[66] S. Toledo, *Improving the memory-system performance of sparse-matrix vector multiplication*, IBM Journal of Research and Development, 41 (1997), pp. 711–725.

[67] H. A. van der Vorst, *Iterative Krylov Methods for Large Linear Systems*, Cambridge University Press, 2003.

[68] R. Vuduc, J. Demmel, K. Yelick, S. Kamil, R. Nishtala, and B. Lee, *Performance optimizations and bounds for sparse matrix-vector multiply*, in Proceedings of Supercomputing, Baltimore, MD, USA, November 2002, 2002.